Service Provisioning Problem in Cloud and multi-Cloud Systems*

Mauro Passacantando† Danilo Ardagna, Anna Savi‡

Abstract. Cloud Computing is a new emerging paradigm that aims at streamlining the on-demand provisioning of resources as services, providing end-user with flexible and scalable services accessible through the Internet on a pay-per-use basis. Since modern Cloud systems operate in an open and dynamic world characterized by continuous changes, the development of efficient resource provisioning policies for Cloud-based services becomes increasingly challenging.

This paper aims to study the hourly basis service provisioning problem through a generalized Nash game model. We take the perspective of SaaS (Software as a Service) providers which want to minimize the costs associated with the virtual machine instances allocated in a multi-IaaSs (Infrastructure as a Service) scenario, while avoiding incurring in penalties for requests execution failures and providing quality of service guarantees. SaaS providers compete and bid for the use of infrastructural resources, while the IaaSs want to maximize their revenues obtained providing virtualized resources.

We propose a solution algorithm based on the best-reply dynamics, which is suitable for a distributed implementation. We demonstrate the effectiveness of our approach by performing numerical tests, considering multiple workloads and system configurations. Results show that our algorithm is scalable and provides significant cost savings with respect to alternative methods (5% on average but up to 260% for individual SaaS providers). Furthermore, varying the number of IaaS providers 8–15% cost savings can be achieved from the workload distribution on multiple IaaSs.

Keywords. Cloud Computing; Game Theory; Generalized Nash Equilibrium.

1 Introduction

Handling workloads of great diversity and enormous scale is necessary in all the most significant fields of today society, due to the penetration of Information and Communications Technology (ICT) in our daily interactions with the world both at personal and community levels, encompassing business, commerce, education, manufacturing, and communication services. With the rapid development of computing and storage technologies, and with the success of the Internet, computing resources have become cheaper, more powerful and more universally available than ever before. In such a setting, dynamic systems are required to provide services and applications that are more competitive, more scalable, and more responsive with respect to traditional systems. This technological trend has enabled the realization of a new computing paradigm called Cloud Computing, in which resources (e.g., CPU and storage) are provided as general utilities that can be leased by users through the Internet in an on-demand fashion.

*This paper has been published in INFORMS Journal of Computing, vol. 28 (2016), pp. 265–277 (see http://dx.doi.org/10.1287/ijoc.2015.0681).

†Department of Computer Science, University of Pisa, Largo B. Pontecorvo 3, 56127 Pisa, Italy, mauro.passacantando@unipi.it

‡Dipartimento di Elettronica, Informazione e Bioingegneria, Politecnico di Milano, Piazza Leonardo da Vinci 32, 20133 Milan, Italy, danilo.ardagna@polimi.it, anna.savi@mail.polimi.it

Typical Cloud-based services exploit the joint effort of an Infrastructure as a Service (IaaS) provider, that manages the Cloud underlying physical resources and is responsible for computational and networking capabilities, and a Software as a Service (SaaS) provider that runs the actual application exploiting the resources of one or more infrastructure providers.

Many companies are offering IaaS solutions such as Amazon Elastic Compute Cloud (EC2) (Amazon Web Services 2014), Google Compute Engine (Google Compute Engine 2014) or Microsoft Windows Azure (Microsoft Windows Azure 2014). Large data centers provide the infrastructure behind the Cloud and virtualization technology makes Cloud computing resources more efficient and cost-effective, while SaaS providers obtain the benefits of the infrastructure without the need to implement and administer it directly. Moreover, SaaS providers can focus on applications development and provide Quality of Service (QoS) guarantees to their end-users, adding or removing capacity almost instantaneously on a "pay-as-you-use" basis.

The growing popularity of Cloud Computing opens new challenges, especially in the area of resource provisioning. Indeed, modern Cloud applications operate in an open and dynamic world characterized by continuous changes which occur autonomously and unpredictably. Moreover, the rapid growth of the Internet and traditional ICT problems, such as resource allocation or QoS, pricing and load shedding, has led to a very complex interaction between all the involved competitors.

In such context, *Noncooperative Game Theory models* have been successfully applied to diverse problems such as Internet pricing, flow and congestion control, routing, and networking (Altman et al. 2006). One of the most widely used solution concept in Game Theory is the *Nash Equilibrium* (Nash 1951): a set of strategies for the players constitute a Nash Equilibrium if no player can benefit by changing his/her strategy unilaterally or, in other words, every player is playing a best response to the strategy choices of his/her opponents.

In this paper, we take the perspective of SaaS providers which host transactional Web applications at multiple IaaS providers. Each SaaS provider wants to minimize the cost of use of Cloud resources and penalties for requests execution failures. The cost minimization is challenging since on-line services receive dynamic workloads changing with the time of the day. Resources have to be allocated flexibly at run-time according to workload variations.

Furthermore, each SaaS behaves selfishly and competes with others SaaS for the use of infrastructural resources supplied by IaaS providers. Each IaaS, in his turn, wants to maximize the revenues obtained providing the resources. To capture the behavior of SaaSs and IaaSs in this conflicting situation, in which the best choice for one depends on the choices of the others, we recur to the *Generalized Nash Equilibrium* (GNE) concept (see e.g., Bigi et al. (2013), Cavazzuti et al. (2002), Debreu (1952), Facchinei and Kanzow (2010a), Rosen (1965)). GNE is an extension of the Nash equilibrium, in which not only the objective function but also the feasible region of each player depend on the strategies chosen by the other players. We propose a solution algorithm based on the best-reply dynamics, which is suitable for a fully distributed implementation. We demonstrate that our algorithm converges to a GNE and the effectiveness of our approach is shown by performing numerical analyses, considering multiple workloads and system configurations, and comparing our solution with other approaches (Wolke and Meixner 2010, Zhu et al. 2009).

In the literature many solutions have been proposed to represent, model, and manage Cloud services at run-time through Game Theory tools. In Feng et al. (2014) authors present an in-depth game theory study on price competition, moving progressively from a monopoly market to a duopoly market, and finally to an oligopoly Cloud market. They characterize the nature of noncooperative competition in a Cloud market with multiple competing Cloud service providers, derive algorithms that represent the influence of resource capacity and operating costs on the solution and they prove the existence of a Nash equilibrium. However, the analysis is performed under very stringent hypothesis (e.g., the whole IaaS data center is modelled as an M/M/1 queue) and Cloud end users cannot split their workload among multiple IaaSs. A model of competitive equilibrium and market dynamics in an e-commerce scenario is proposed in Dube et al. (2007).

Here the authors analyse pricing choices and decisions to outsource ICT systems providing a representation of the Internet competition and the solution maximizing profits for two players. A recent survey on Cloud service pricing models is provided also in Gohad et al. (2013).

Studies on the maximization of the social welfare as a long-term social utility are discussed in Menache et al. (2011). Under appropriate convexity assumptions on the operating costs and individual utilities, the work established the existence and uniqueness of the social optimum, considering relevant queuing aspects in a centralized setting. Moreover, the study is performed under the assumptions that the system resources are sufficient to fully accomodate the social optimal demand (while, our GNE technique is more general and especially suitable to model resource congestion).

Other studies are presented in Wan et al. (2012), where authors employ a bidding model to solve the resource allocation problem in virtualized servers with multiple virtual machine instances competing for physical resources. A unique equilibrium point is obtained. However, the model considers CPU utilization as the main resource allocation metric, without providing any response time guarantees to the running applications. A similar discussion can be found in Wei et al. (2010) where a QoS constrained parallel tasks resource allocation problem is considered. In particular, a task to physical machine assignment problem is analysed. Authors proposed a distributed solution, which initially solves the task assignment problem assuming that the available cluster can be dedicated to each task execution. Resource contention is then considered and the final solution, demonstrated to be a Nash equilibrium, is obtained through an evolutionary optimization process. In Abhishek et al. (2012) the authors consider two simple pricing schemes for selling Cloud instances and study the trade-off between them. Exploiting Bayesian Nash equilibrium the authors provide theoretical and simulation based evidence suggesting that fixed prices generate a higher expected revenue than hybrid systems. However, a single IaaS system and only two job classes are considered. Using Bellman equations and a dynamic bidding policy in Zafer et al. (2012), an optimal strategy for a single user under a Markov spot price evaluation is found to serve jobs with deadline and availability constraints. User's requests are modelled in terms of overall CPU hours that need to be executed within a given deadline. Another work regarding on-spot bidding is proposed in Song et al. (2012). Authors propose a profit aware dynamic bidding algorithm, which observes the current spot price and selects bids adaptively to maximize the average profit of a Cloud service broker while minimizing its costs in a spot instance market.

In Ardagna et al. (2011, 2013), Anselmi et al. (2014) we used the GNE concept for a service provisioning problem where the perspective of SaaS providers hosting their applications at a single provider is taken. Each SaaS needs to comply with end user applications Service Level Agreement (SLA) and, at the same time, maximize its own revenue, while minimizing the cost of use of resources supplied by the IaaS. On the other hand, the IaaS wants to maximize the revenues obtained providing on spot resources. With respect to our previous works, in this paper we consider the allocation of SaaS resources on multiple IaaS providers which is far more challenging and complex with respect to the allocation of resources on a single Cloud. Furthermore, more realistic pricing models are considered which lead to nonconvex feasible sets for which current literature results cannot guarantee even the GNE existence. To the best of our knowledge, the only work considering a multi-SaaS and multi-IaaS scenario is presented in Roh et al. (2013), where a resource pricing problem in geo-distributed Cloud is presented. Authors propose a Stackelberg game-theoretic framework that is further reduced to a Rosen's concave game. However, more restrictive assumptions than our work are introduced (i.e., multiple VMs running at a Cloud data center are modelled as a single M/M/1 queue and the strategy space of the resource pricing game is a convex set) and on spot resources are not considered.

With respect to previous literature proposals the main contributions of our paper include: (1) we consider a realistic pricing model for on-spot resource market that is currently used by Cloud providers, (2) we model Multiple SaaS-Multiple IaaS competition, (3) we provide an efficient algorithm suitable for a distributed implementation and demonstrate that our solution reaches a GNE under mild hypotheses.

Finally, our numerical analyses show that our algorithm is scalable and provides significant cost savings

with respect to alternative methods currently adopted in real systems (Wolke and Meixner 2010, Zhu et al. 2009) (5% on average but up to 260% for individual SaaS providers can be obtained). Furthermore, by varying the number of IaaS providers we show that 8-15% cost savings can be achieved from the workload distribution on multiple IaaSs.

The remainder of the paper is organized as follows. Section 2 describes the problem under study and introduces the design assumptions. In Section 3 the SaaS and IaaS problems are formalized and the service provisioning problem is modelled as a Generalized Nash Equilibrium Problem. Then, a distributed algorithm based on the best-reply dynamics is provided in Section 4 in order to find a GNE which is efficient from the SaaS providers point of view. The experimental results are discussed in Section 5. Conclusions and some future research directions are finally drawn in Section 6.

2 Problem Statement

Our model considers SaaS providers using Cloud Computing facilities according to the IaaS paradigm to offer multiple transactional Web services (WSs), each service representing a different application.

The hosted WSs can be heterogeneous with respect to resource demands, workload intensities and QoS requirements. The set of IaaS will be indicated as $\mathcal{I}$, while $\mathcal{S}$ will indicate the set of SaaSs. $\mathcal{S}_i$ denotes the set of SaaS providers running at the IaaS provider i and $\mathcal{I}_j$ the set of IaaS providers supporting SaaS j. The set of WS applications offered by the j-th SaaS provider is denoted by $\mathcal{A}_j$; the set of applications running at IaaS i is denoted by $\mathcal{A}_i$ and $\mathcal{A}$ is the set of applications of all the SaaS providers.

An SLA contract, associated with each WS application, is established between the SaaS provider and its end-users. Among the possible SLAs (see, e.g., Urgaonkar et al. (2007)), we assume that an average response time needs to be guaranteed for WS applications and we will indicate with $\overline{R}_k$ the response time threshold for the execution of each WS application, i.e., $E[R_k] \leq \overline{R}_k$ for all $k \in \mathcal{A}_j$. Furthermore, SaaS providers implement an admission control mechanism (Almeida et al. 2010) and can reject requests under heavy loads. According to the SLA, if the SaaS provider rejects a request, the SLA is violated and the SaaS incurs in penalties; we will denote with ν_k the penalty for rejecting a single application k request. However, in order to limit the admission control mechanism (a rejection for an end user is equivalent to unavailability of service) a minimum throughput λ_k is guaranteed for every $k \in \mathcal{A}_j$.

Applications are hosted in virtual machines (VMs) which are dynamically instantiated by the IaaS providers up to a maximum number of N_i for each IaaS i. For the sake of simplicity, we have imposed that VMs are homogeneous providing a maximum service rate μ_{ki} for the requests of application k running at the IaaS i, but this constraint can be easily relaxed[1]. In the following, we assume that SaaS providers host their application at multiple IaaS providers that compete each other in the Cloud market. The interoperability among, possibly, heterogenous technologies is guaranteed by the middleware layer developed by the MODAClouds project (MODAClouds 2014, Ardagna et al. 2012b) which allows to run applications concurrently on multiple Clouds increasing the availability of the whole system. Furthermore, we make the simplifying assumption that each VM hosts a single WS application.

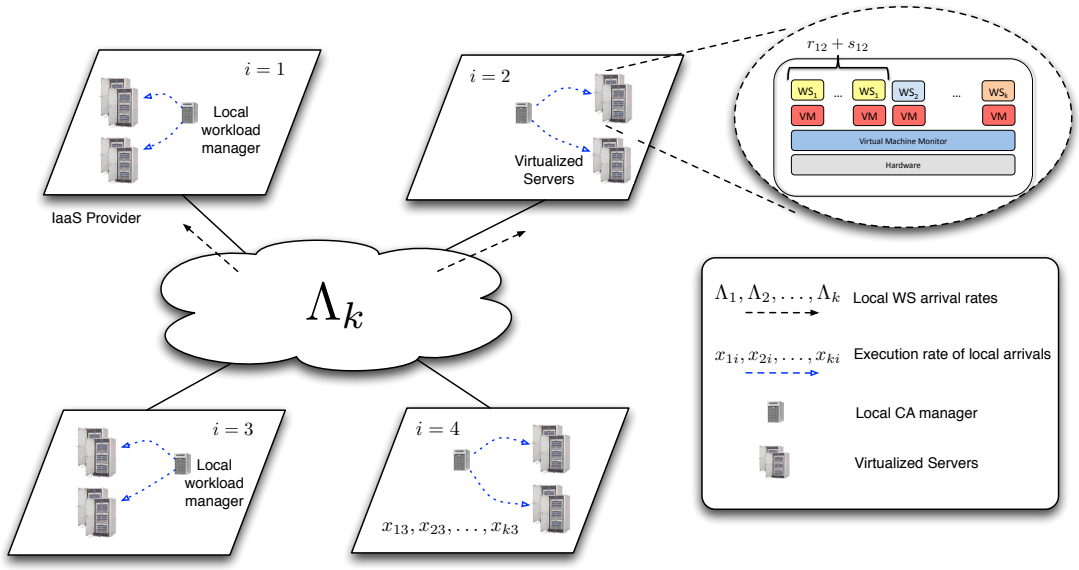
IaaS providers usually charge the use of their resources on an hourly basis. Hence, the SaaS faces the problem of determining every hour the optimal number of VMs for each WS class in order to minimize costs and penalties. Resource allocation is performed on the basis of a prediction of future WS workloads (Ardagna et al. 2012a, Zhu et al. 2009) and we will denote with Λ_k the prediction of the arrival rate for WS application k for the next time horizon. The SaaS needs also an estimate of the future performance of each VM in order

[1]Running homogeneous VMs in IaaS is today common practice (Amazon Inc. 2015b). In case of heterogeneous VMs, our approach can be extended by implementing the proportional assignment schema, which assigns the incoming workload to running VMs proportionally to their capacity. See Ardagna et al. (2011) for further details.

to determine application average response time. We model, as a first approximation and as in the other literature approaches (see, e.g., Kumar et al. (2009), Zhang et al. (2013)), each WS application hosted in a VM as an M/G/1 queue in tandem with a delay center, under the assumption that it is CPU bounded. We assume (as common among Web service containers) that requests are served according to the processor sharing scheduling discipline. As discussed in Ardagna et al. (2013), the delay center allows to model network delays and/or protocol delays introduced in establishing connections, etc. and it will be denoted with D_{ki}. Performance parameters are also continuously updated at run-time in order to capture transient behaviours, VMs network and I/O interference and performance time of the day variability of the Cloud provider, as in Zhang et al. (2013).

Multiple VMs can run in parallel to support the same application. In that case, we suppose that the workload is evenly shared among multiple instances (see Figure 1), which is common for current Cloud solutions (Amazon Elastic Cloud 2014).

Figure 1: Cloud Infrastructures.



For IaaS providers we consider a pricing model similar to Amazon EC2. Each IaaS provider offers: (i) *reserved* VMs, for which SaaS providers applies for a one-time payment (currently every one or three years) for each instance they want to reserve, and (ii) *on spot* VMs, for which SaaS providers bid and compete for unused IaaS capacity. We will denote with r_{ki} and s_{ki} the number of reserved and on spot instances supporting the WS application k at IaaS i, respectively. Moreover, R_{ij} denotes the maximum number of reserved VMs available for SaaS j at IaaS i.

The VM instances are charged with the reserved cost ρ_i and on spot cost σ_i for each SaaS hosted at IaaS i. These latter costs are set by the IaaS and fluctuate periodically depending on the IaaS provider time of the day energy costs ω_i and also on the supply and demand from SaaS for on spot VM. Indeed, each SaaS provider j competes for the use of on spot VMs specifying the maximum cost $\overline{\sigma}_{ij}$ it is willing to pay per instance per hour at IaaS i and the number of on spot VMs it wants to use $\overline{s}_{ki}$ for application k at IaaS i. If IaaS i sets the on spot cost σ_i less or equal to the threshold $\overline{\sigma}_{ij}$, then the SaaS j obtains the use of s_{ki} ($\leq \overline{s}_{ki}$) on spot VMs that IaaS i dedicates for application k, otherwise the IaaS decides not to allocate any on spot instance to SaaS j.

IaaS provider i has to determine every hour the time unit cost σ_i for on spot instances and the number s_{ki} of on spot instances to be allocated to application k, in order to maximize its total revenue.

On the other side, SaaS provider j has to determine every hour[2]: the throughput x_{ki} for the execution of the WS application k at site i; the number r_{ki} of reserved VMs supporting the WS application k at IaaS i; the number $\overline{s}_{ki}$ of desired on spot VMs supporting the WS application k at IaaS i; the maximum cost $\overline{\sigma}_{ij}$ it is willing to pay for on spot VMs instances, in order to minimize both the costs of reserved and on spot VMs and the penalties due to request rejections.

Since on spot resource are less reliable than reserved ones, the price $\overline{\sigma}_{ij}$ offered by SaaS j to IaaS i is assumed to be lower than a fraction $q_{ij} \in (0,1)$ of the reserved VM cost ρ_i. The values q_{ij} depend on how much the SaaS j is willing to pay for a unreliable resource. Moreover, in order to guarantee a minimum reliability level for every WS application, the number of on spot VMs required by SaaS j to any IaaS provider for any WS application is assumed to be lower than a fraction $\eta_j \in (0,1)$ of the total number of resources requested. Indeed, if only on spot VMs are adopted by the SaaS and they are terminated by the IaaS, an application could become unavailable (however, we assume that on spot VMs could become unavailable only for SaaS competition to access IaaS resources, neglecting server disruptions, data center power breakdowns, etc.).

We assume that the SaaS decisions are taken according to some i.i.d. probabilistic law. The resulting application execution rate (or throughput, acceptance rate) is denoted by $X_k = \sum_{i \in \mathcal{I}_j} x_{ki}$ for application k and is less or equal to the prediction Λ_k for the arrival rate of the WS application k. If the workload is evenly shared among the VMs, then the average response time for execution of application k requests is given by:

$$E[R_{ki}] = D_{ki} + \frac{1}{\mu_{ki} - \left(\frac{x_{ki}}{r_{ki}+s_{ki}}\right)}, \tag{1}$$

under the assumption that the VMs are not saturated (i.e., the equilibrium conditions for the M/G/1 queues hold, $\mu_{ki}(r_{ki} + s_{ki}) - x_{ki} > 0$).

For the sake of clarity, the notation adopted here is summarized in Table 1.

3 Generalized Nash Game Model

The resource provisioning problem for the Cloud Computing system under study describes a conflicting situation, in which the optimal choices of SaaS and IaaS providers depend on the choices of the others. In this section we formulate this problem as a Generalized Nash Equilibrium Problem (GNEP): Section 3.1 is devoted to the formulation of the SaaS resource allocation problem, Section 3.2 contains the IaaS providers optimization problems and the Generalized Nash equilibria of the game are defined in Section 3.3.

3.1 Game Formulation from the SaaS Side

The problem that SaaS provider j has to periodically solve can be formulated as follows:

[2]In our paper we assume SaaS capacity allocation is performed periodically according to one hour workload prediction. This assumption can be too restrictive under fast varying workloads. In such situations, or when QoS constraints become violated, to react to unexpected events (e.g., unexpected peaks or VM failures), we assume that SaaSs perform capacity allocation, possibly considering a restricted set of decision variables, at a more fine-grain time scale (e.g., 5 or 10 min) on the basis of a short-term prediction of future WS workloads (Menasc and Bennani 2006). However, the analysis of capacity allocation re-optimization at finer grained time scales is out of the scope of this paper.

Table 1: Parameters and decision variables.

System Parameters

$\mathcal{S}$	Set of SaaS providers
$\mathcal{I}$	Set of IaaS providers
$\mathcal{S}_i$	Set of SaaS providers running applications at IaaS i
$\mathcal{I}_j$	Set of IaaS providers supporting SaaS j
$\mathcal{A}$	Set of applications of all the SaaS providers
$\mathcal{A}_j$	Set of applications of the SaaS provider j
$\mathcal{A}_i$	Set of applications running at IaaS i
Λ_k	Prediction of the arrival rate for application k
λ_k	Minimum arrival rate to be guaranteed for application k
μ_{ki}	Maximum service rate for executing class k application at IaaS i
D_{ki}	Queueing delay for executing class k application at IaaS i
$\overline{R}_k$	Application k average response time threshold
ν_k	Penalty for rejecting a single application k request
ρ_i	Time unit cost for reserved VMs at IaaS i
q_{ij}	Maximum fraction of reserved VMs price for on spot VMs price SaaS provider j is willing to pay to IaaS i
ω_i	VM time unit energy cost for IaaS provider i
η_j	Maximum fraction of total resources allocated as on spot VMs for SaaS provider j
N_i	Maximum number of VMs that can be executed at the IaaS i
R_{ij}	Maximum number of reserved VMs that can be executed for the SaaS j at IaaS i
T	Control time horizon

SaaS Decision Variables

r_{ki}	Number of reserved VMs used for application k at IaaS i
$\overline{s}_{ki}$	Number of desired on spot VMs for application k at IaaS site i
x_{ki}	Throughput for application k at IaaS i
X_k	Overall throughput for application k
$\overline{\sigma}_{ij}$	Time unit cost threshold for SaaS j for on spot VM instances at site i

IaaS Decision Variables

s_{ki}	Number of on spot VMs used for application k at IaaS site i
σ_i	Time unit cost offered for on spot VM instances at site i

$$\min_{r_{ki},\overline{s}_{ki},x_{ki},X_k,\overline{\sigma}_{ij}} \Theta_j = \sum_{k\in\mathcal{A}_j}\sum_{i\in\mathcal{I}_j}(\rho_i\, r_{ki} + \sigma_i\, s_{ki}) + \sum_{k\in\mathcal{A}_j} T\,\nu_k\,(\Lambda_k - X_k) \tag{2}$$

subject to:

$$D_{ki} + \frac{1}{\mu_{ki} - \left(\frac{x_{ki}}{r_{ki}+s_{ki}}\right)} \leq \overline{R}_k \qquad \forall\ k \in \mathcal{A}_j,\ \forall\ i \in \mathcal{I}_j, \tag{3}$$

$$\sum_{i\in\mathcal{I}_j} x_{ki} = X_k \qquad \forall\ k \in \mathcal{A}_j, \tag{4}$$

$$\lambda_k \leq X_k \leq \Lambda_k \qquad \forall\ k \in \mathcal{A}_j, \tag{5}$$

$$\sum_{k\in\mathcal{A}_j} r_{ki} \leq R_{ij} \qquad \forall\ i \in \mathcal{I}_j, \tag{6}$$

$$\overline{s}_{ki} \leq \frac{\eta_j}{1-\eta_j} r_{ki} \qquad \forall\ k \in \mathcal{A}_j,\ \forall\ i \in \mathcal{I}_j, \tag{7}$$

$$\overline{\sigma}_{ij} \leq q_{ij}\,\rho_i \qquad \forall\ i \in \mathcal{I}_j, \tag{8}$$

$$r_{ki},\ \overline{s}_{ki},\ x_{ki},\ \overline{\sigma}_{ij} \geq 0 \qquad \forall\ k \in \mathcal{A}_j,\ \forall\ i \in \mathcal{I}_j. \tag{9}$$

The first two terms of the objective function represent the costs of reserved and on spot VMs, respectively, while the third term determine the penalties incurred with request rejections. Constraint (3) ensures that the average response time $E[R_{ki}]$ (see equation (1)) is less or equal to the threshold $\overline{R}_k$ established in the SLA contract. Note that (3) can be equivalently rewritten as a linear constraint in terms of variables r_{ki} and x_{ki}. Constraints (4) and (5) represent the bounds on the throughput X_k: the lower bound λ_k is needed to satisfy SLA contracts and the upper bound Λ_k is the application request rate prediction. Constraint (6) entails that the reserved VMs allocated to IaaS i are less or equal to the maximum number available R_{ij}. Constraint (7) is introduced for fault tolerance reasons, as explained before, and guarantees that the number of on spot instances $\overline{s}_{ki}$ is at most a fraction η_j of the total number $r_{ki} + \overline{s}_{ki}$ of VMs requested to IaaS i for application k. Constraint (8) sets the on spot price $\overline{\sigma}_{ij}$ offered by SaaS j to IaaS i less or equal to a fraction q_{ij} of the reserved VM cost ρ_i.

We remark that we have imposed variables s_{ki} and r_{ki} to be continuous but not integer, as in reality they are. In fact, requiring variables to be integer could make the solution much more difficult from the computational perspective. We therefore decide, as in several literature approaches (see, e.g., Ardagna et al. (2013), Zhang et al. (2012)), to deal with continuous variables, actually considering the continuous relaxation of the problem. However, experimental results have shown that if the fractional optimal values of the variables are rounded to the closest integer solution, the gap between the optimal value of the integer problem and the optimal value of the relaxed one is very small. This is intuitive for large scale data centers including thousands of servers and is a common assumption adopted in the literature (Zhang et al. 2012).

The SaaS problem is a Linear Programming problem which can be solved very efficiently by state of the art solvers.[3]

3.2 Game Formulation from the IaaS Side

Every IaaS provider i has to solve the following optimization problem:

[3]As discussed in the previous section, in case of QoS violation or unexpected events, each SaaS can independently solve a restricted problem at finer grained time scales, neglecting $\overline{s}_{ik}$ decision variables and possibly relaxing $X_k \geq \lambda_k$ constraint. However, the analysis of SaaS problem solution at finer grained time scales is out of the scope of this paper.

$$\max_{s_{ki}, y_{ij}, \sigma_i} \Theta_i = \sum_{k \in \mathcal{A}_i} \left[(\rho_i - \omega_i)\ r_{ki} + (\sigma_i - \omega_i)\ s_{ki} \right] \tag{10}$$

subject to:

$$\sum_{k \in \mathcal{A}_i} s_{ki} \leq N_i - \sum_{k \in \mathcal{A}_i} r_{ki}, \tag{11}$$

$$\sigma_i \geq \omega_i, \tag{12}$$

$$\sigma_i - \overline{\sigma}_{ij} \leq M\,(1 - y_{ij}) \qquad \forall\ j \in \mathcal{S}_i, \tag{13}$$

$$\overline{\sigma}_{ij} - \sigma_i \leq M\,y_{ij} \qquad \forall\ j \in \mathcal{S}_i, \tag{14}$$

$$0 \leq s_{ki} \leq \overline{s}_{ki}\,y_{ij} \qquad \forall\ j \in \mathcal{S}_i,\ \forall\ k \in \mathcal{A}_j, \tag{15}$$

$$y_{ij} \in \{0, 1\} \qquad \forall\ j \in \mathcal{S}_i. \tag{16}$$

The two terms of the objective function represent the IaaS i profit obtained by reserved and on spot VMs, respectively. Constraint (11) entails that the total number of on spot VMs allocated to applications is less or equal to the difference between the maximum number N_i of VMs that can be executed at the IaaS i and the total number of reserved VMs allocated to applications. Constraint (12) guarantees that the on spot instance cost σ_i is at least equal to the energy cost ω_i. Constraints (13)– (15) model the dynamics between SaaS providers and IaaS i about the cost and the number of on spot VMs. In fact, exploiting the auxiliary binary variable y_{ij}, we obtain that if IaaS i chooses the on spot cost σ_i greater than $\overline{\sigma}_{ij}$, then (13) implies $y_{ij} = 0$ and (15) gives $s_{ki} = 0$ for all applications $k \in \mathcal{A}_j$, i.e., IaaS i decides not to allocate any on spot instance to SaaS j. On the other hand, if $\sigma_i < \overline{\sigma}_{ij}$, then (14) implies $y_{ij} = 1$ and (15) gives $s_{ki} \leq \overline{s}_{ki}$, i.e., the SaaS j obtains the use of s_{ki} on spot VMs which is less or equal to the required number $\overline{s}_{ki}$.

The IaaS optimization problem stated above is a Mixed Integer Nonlinear Programming problem, in which the objective function is neither convex nor concave (because it contains the products $\sigma_i\,s_{ki}$), thus it is very challenging from a computational point of view. However, we now show that its special structure allows to develop a simple *ad-hoc* algorithm for finding a global optimal solution.

For the sake of simplicity, let us assume that the on spot prices $\overline{\sigma}_{ij}$ offered by the SaaS providers are decreasingly ordered, i.e., $\overline{\sigma}_{i1} > \overline{\sigma}_{i2} > \cdots > \overline{\sigma}_{i|\mathcal{S}_i|}$. The idea of the solution algorithm is based on the following argument: if IaaS chooses $\sigma_i > \overline{\sigma}_{i1}$, then its revenue is null; if $\sigma_i = \overline{\sigma}_{i1}$, then it offers on spot VMs only to the first SaaS, obtaining the revenue $\mathcal{R}_1 = \overline{\sigma}_{i1} \sum_{k \in \mathcal{A}_1} \overline{s}_{ki}$ (provided that the number of on spot VMs available is at least $\sum_{k \in \mathcal{A}_1} \overline{s}_{ki}$); if IaaS sets $\sigma_i \in (\overline{\sigma}_{i2}, \overline{\sigma}_{i1})$, then it offers again on spot VMs only to the first SaaS, but its revenue is $\sigma_{i1} \sum_{k \in \mathcal{A}_1} \overline{s}_{ki}$, which is lower than $\mathcal{R}_1$. Next, if the IaaS reduces σ_i to $\overline{\sigma}_{i2}$, this time it gains $\mathcal{R}_2 = \overline{\sigma}_{i2} \sum_{k \in \mathcal{A}_1 \cup \mathcal{A}_2} \overline{s}_{ki}$; if it sets $\sigma_i \in (\overline{\sigma}_{i3}, \overline{\sigma}_{i2})$, then again this choice is not convenient since the revenue is lower than $\mathcal{R}_2$, and so on. Therefore, the optimal price σ_i must be equal to one of the on spot prices $\overline{\sigma}_{ij}$ offered by the SaaS providers and it can be obtained by iterating this process until there are no more VMs available according to the capacity constraint (11). The complete method to obtain the global optimum solution of IaaS problem is reported in Algorithm 3.1.

The algorithm starts sorting the prices $\overline{\sigma}_{ij}$ offered by the SaaSs and determines the number S_i of on spot VMs that the IaaS i can offer (steps 1 and 2). During the execution of the algorithm σ_i represents the current best price for the IaaS, $\mathcal{R}_i$ the corresponding revenue and t is an index to iterate among SaaSs. These three values are initialized in step 3. Steps 4–8 find the optimal cost σ_i. The IaaS determines if there is enough capacity to provide to the first t SaaSs the on spot VMs at the price $\overline{\sigma}_{it}$ offered by the SaaS t. The maximum number s of on spot VMs that the IaaS can sell to the first t SaaSs is the minimum

Algorithm 3.1: Solving the IaaS i optimization problem.

1 Sort $\overline{\sigma}_{ij}$ in decreasing order: $\overline{\sigma}_{i1} > \overline{\sigma}_{i2} > \cdots > \overline{\sigma}_{i|\mathcal{S}_i|}$
2 $S_i = N_i - \sum_{k \in \mathcal{A}_i} r_{ki}$
3 $\sigma_i = \infty,\ \mathcal{R}_i = 0,\ t = 1$
4 $s = \min \left\{ S_i, \sum_{1 \le j \le t} \sum_{k \in \mathcal{A}_j} \overline{s}_{ki} \right\},\ \mathcal{R} = \overline{\sigma}_{it}\, s$
5 **if** $\mathcal{R} > \mathcal{R}_i$ **then**
6 $\sigma_i = \overline{\sigma}_{it},\ \mathcal{R}_i = \mathcal{R}$
7 **if** *($s < S_i$ and $t < |\mathcal{S}_i|$)* **then**
8 $t = t + 1$, go to step 4
9 **for** $j \in \mathcal{S}_i$ **do**
10 **if** $\overline{\sigma}_{ij} \ge \sigma_i$ **then**
11 $y_{ij} = 1$
12 **for** $k \in \mathcal{A}_j$ **do**
13 $s_{ki} = \min\{\overline{s}_{ki},\ S_i\}$, $S_i = S_i - s_{ki}$
14 **else**
15 $y_{ij} = 0$
16 $s_{ki} = 0 \quad \forall\ k \in \mathcal{A}_j$

between the available capacity S_i and the total number of on spot VMs requested by the first t SaaSs, that is $\sum_{1 \le j \le t} \sum_{k \in \mathcal{A}_j} \overline{s}_{ki}$. The corresponding revenue is $\mathcal{R} = \overline{\sigma}_{it}\, s$ (step 4). Then, in steps 5 and 6, σ_i and $\mathcal{R}_i$ are updated accordingly. If the IaaS capacity is not saturated ($s < S_i$) and there are still SaaS providers to consider ($t < |\mathcal{S}_i|$), then steps 4–6 are repeated; otherwise σ_i is the optimal cost (steps 7 and 8). The optimal values of y_{ij} and s_{ki} are assigned in steps 9–16 according to the price offered by SaaS providers and their WS applications requirements.

Notice that the time complexity of Algorithm 3.1 is $\mathcal{O}\left(\max\left\{|\mathcal{S}_i| \log(|\mathcal{S}_i|),\ |\mathcal{A}_i|\right\}\right)$, due to the sorting at step 1 and the assignment of s_{ki} at step 13.

3.3 Generalized Nash Equilibria

In this framework, SaaS providers and the IaaS providers are taking decisions at the same time. The objective function of each SaaS (IaaS) depends on the variables of the IaaSs (SaaSs). Moreover, the strategy set each SaaS (IaaS) depends on the variables of the IaaSs (SaaSs). In this setting, we can not analyse decision in isolation, but we must ask what a SaaS would do, taking into account the decision of the IaaSs and other SaaSs. To capture the behavior of SaaSs and IaaSs in this conflicting situation (*game*) in which what a SaaS or a IaaS (the *players* of the game) does directly affects what others do, we consider the Generalized Nash game, which is broadly used in Game Theory and other fields. We remind the reader that the GNEP differs from the classical Nash Equilibrium Problem since, not only the objective function of each player depends upon the strategies chosen by all the other players, but also the strategy set of each player may depend on the rival players' strategies.

The service provisioning problem results in a GNEP where the strategies of SaaS j are $r_j = (r_{ki})_{k \in \mathcal{A}_j, i \in \mathcal{I}_j}$, $\overline{s}_j = (\overline{s}_{ki})_{k \in \mathcal{A}_j, i \in \mathcal{I}_j}$, $x_j = (x_{ki})_{k \in \mathcal{A}_j, i \in \mathcal{I}_j}$ and $\overline{\sigma}_j = (\overline{\sigma}_{ij})_{i \in \mathcal{I}_j}$ and , while the strategies of IaaS i are $s_i = (s_{ki})_{k \in \mathcal{A}_i}$, $y_i = (y_{ij})_{j \in \mathcal{S}_i}$ and σ_i.

In this setting, a Generalized Nash Equilibrium (GNE) is a set of strategies such that no player can improve its payoff function by changing its strategy unilaterally (Facchinei and Kanzow 2010a), i.e., a GNE is a vector $(r^*, \overline{s}^*, x^*, \overline{\sigma}^*, s^*, y^*, \sigma^*)$ such that constraints (3)–(9) and (11)–(16) are satisfied, for any $j \in \mathcal{S}$ we have

$$\Theta_j(r_j^*, x_j^*, s^*, \sigma^*) \leq \Theta_j(r_j, x_j, s^*, \sigma^*), \qquad \forall\ (r_j, x_j) \text{ satisfying constraints (3)–(9)}, \tag{17}$$

and for all $i \in \mathcal{I}$ we have

$$\Theta_i(r^*, s_i^*, \sigma_i^*) \geq \Theta_i(r^*, s_i, \sigma_i), \qquad \forall\ (s_i, \sigma_i) \text{ satisfying constraints (11)–(16)}. \tag{18}$$

4 Solution Algorithm

In the last years, several algorithms for solving GNEPs have been proposed in the literature (Facchinei and Kanzow 2010b, von Heusinger and Kanzow 2009, von Heusinger et al. 2012, Nabetani et al. 2011, Pang and Fukushima 2005, Panicucci et al. 2009). All these approaches can be applied to GNEPs in which each player has to solve a convex optimization problem. Moreover, most of them solves GNEPs with shared constraints. Unfortunately, the GNEP stated in the previous section does not meet any of the two conditions. In fact, the IaaSs optimization problems are not convex since involve binary variables y_{ij} and the constraints of each player are not shared with all the other players. Therefore, the GNEP considered in this paper is very challenging to solve. In this section, we propose an *ad-hoc* algorithm based on the best-reply dynamics, which is suitable for a distributed implementation, in order to find a GNE. The complete procedure is reported in Algorithm 4.1.

Steps 1–10 initialize the values of SaaSs and IaaSs variables. The algorithm starts (step 1) choosing, for any IaaS i, the on spot price σ_i as 10% higher than the energy costs ω_i. Next, each SaaS j computes the optimal value of reserved and on spot resources considering the ideal scenario in which the overall Cloud system workload is light and all the necessary resources can be obtained from IaaSs (step 3). In other words, in the ideal scenario we assume that $s_{ki} = \overline{s}_{ki}$ for each WS application $k \in \mathcal{A}_j$ and each IaaS $i \in \mathcal{I}_j$, and there is no SaaSs competition on the on spot prices. The problem of SaaS j in the ideal scenario, denoted by `IdealSaaS[j]`, can be formulated as follows:

$$\min_{r_{ki}, \overline{s}_{ki}, x_{ki}} \sum_{k \in \mathcal{A}_j} \sum_{i \in \mathcal{I}_j} (\rho_i\, r_{ki} + \sigma_i\, \overline{s}_{ki}) + \sum_{k \in \mathcal{A}_j} T\, \nu_k\ (\Lambda_k - X_k) \tag{19}$$

subject to:

$$D_{ki} + \frac{1}{\mu_{ki} - \left(\frac{x_{ki}}{r_{ki} + \overline{s}_{ki}}\right)} \leq \overline{R}_k \qquad \forall\ k \in \mathcal{A}_j,\ \forall\ i \in \mathcal{I}_j, \tag{20}$$

$$\sum_{i \in \mathcal{I}_j} x_{ki} = X_k \qquad \forall\ k \in \mathcal{A}_j, \tag{21}$$

$$\lambda_k \leq X_k \leq \Lambda_k \qquad \forall\ k \in \mathcal{A}_j, \tag{22}$$

$$\sum_{k \in \mathcal{A}_j} r_{ki} \leq R_{ij} \qquad \forall\ i \in \mathcal{I}_j, \tag{23}$$

$$r_{ki}, \overline{s}_{ki}, x_{ki} \geq 0 \qquad \forall\ k \in \mathcal{A}_j,\ \forall\ i \in \mathcal{I}_j. \tag{24}$$

Moreover, each SaaS j sets the on spot price thresholds $\overline{\sigma}_{ij}$ as the minimum between σ_i set by IaaSs and the fraction q_{ij} of reserved price ρ_i (step 4). Then, each IaaS i assigns the on spot resources to SaaSs: it

Algorithm 4.1: Solution algorithm

$\sigma_i = 1.1\,\omega_i, \quad \forall\ i \in \mathcal{I}$

for $j \in \mathcal{S}$ **do**

  Solve `IdealSaaS[j]`

  $\overline{\sigma}_{ij} = \min\{\sigma_i,\ q_{ij}\,\rho_i\}, \quad \forall\ i \in \mathcal{I}_j$

for $i \in \mathcal{I}$ **do**

  for $j \in \mathcal{S}_i$ *s.t.* $\overline{\sigma}_{ij} < \sigma_i$ **do**

    $s_{ki} = 0, \quad \forall\ k \in \mathcal{A}_j$

  $S_i^{tot} = \sum_{\substack{j \in \mathcal{S}_i \\ \text{s.t. } \overline{\sigma}_{ij} = \sigma_i}} \sum_{k \in \mathcal{A}_j} \overline{s}_{ki}$

  for $j \in \mathcal{S}_i$ *s.t.* $\overline{\sigma}_{ij} = \sigma_i$ **do**

    $s_{ki} = \overline{s}_{ki}\ \min\left\{1,\ \dfrac{N_i - \sum_{k \in \mathcal{A}_i} r_{ki}}{S_i^{tot}}\right\}, \quad \forall\ k \in \mathcal{A}_j$

continue = 0

for $j \in \mathcal{S}$ **do**

  solve `SaaS[j]`

  if *(problem is unfeasible) and* $(\exists\ i \in I_j$ *s.t.* $\overline{\sigma}_{ij} \le \sigma_i < q_{ij}\,\rho_i)$ **then**

    solve `IdealSaas[j]`

    for $i \in \mathcal{I}_j$ *s.t.* $\overline{\sigma}_{ij} \le \sigma_i < q_{ij}\,\rho_i$ **do**

      $\overline{\sigma}_{ij} = \min\{1.01\sigma_i,\ q_{ij}\,\rho_i\}$

    continue = 1

  else

    if *(*$\exists\ k \in \mathcal{A}_j$ *s.t.* $X_k < \Lambda_k$*) and* $(\exists\ i \in I_j$ *s.t.* $\overline{\sigma}_{ij} \le \sigma_i < q_{ij}\,\rho_i)$ **then**

      for $i \in \mathcal{I}_j$ *s.t.* $\overline{\sigma}_{ij} \le \sigma_i < q_{ij}\,\rho_i$ **do**

        $\overline{\sigma}_{ij} = \min\{1.01\sigma_i,\ q_{ij}\,\rho_i\}$

    continue = 1

for $i \in \mathcal{I}$ **do**

  solve `IaaS[i]`

if *continue* = 1 **then**

  go to step 11

decides not to allocate any on spot instance to each SaaS j offering a price $\overline{\sigma}_{ij}$ lower than σ_i (steps 6–7). It computes the total number S_i^{tot} of on spot VMs required by SaaSs j with $\overline{\sigma}_{ij} = \sigma_i$ (step 8); if S_i^{tot} is less or equal to the total number of available on spot VMs, i.e., $N_i - \sum_{k \in \mathcal{A}_i} r_{ki}$, then it assigns $s_{ki} = \overline{s}_{ki}$ for all $k \in \mathcal{A}_j$; otherwise s_{ki} are rescaled proportionally to the ratio $\left(N_i - \sum_{k \in \mathcal{A}_i} r_{ki} \right) / S_i^{tot}$ (step 10).

Subsequently, the best-reply dynamics starts. Each SaaS j solves its own problem `SaaS[j]`; if this problem is unfeasible, i.e., SaaS j cannot guarantee the minimum throughput λ_k for some $k \in \mathcal{A}_j$, and there exists at least a IaaS from which it could obtain more on spot resources, then it solves the problem in the ideal scenario (step 15) and it increases the on spot price threshold $\overline{\sigma}_{ij}$ up to 1% higher than the on spot price σ_i, provided that it is below the bound $q_{ij}\,\rho_i$ (steps 15–17). This increase of $\overline{\sigma}_{ij}$ is performed also in the case some workload is rejected for some class $k \in \mathcal{A}_j$, i.e., $X_k < \Lambda_k$ (steps 20–22). We adopt the flag `continue` to indicate that some SaaS has increased the on spot price threshold $\overline{\sigma}_{ij}$ and it needs to play again. Next, each IaaS solves its own problem `IaaS[i]` (steps 24–25). If the flag `continue` is equal to 1, then all the players need to solve again their problems, otherwise the strategies of each player are the best response to the ones of the other players, that is a GNE is obtained. We can demonstrate the following important result:

Theorem 4.1. *Algorithm 4.1 finds a Generalized Nash Equilibrium after a finite number of iterations.*

Proof. If the flag `continue` is equal to 1 at step 26, then there exists at least a SaaS provider j which increased the on spot price threshold $\overline{\sigma}_{ij}$ at least by 1% or set $\overline{\sigma}_{ij}$ equal to the maximum possible value $q_{ij}\,\rho_i$. Since the values $\overline{\sigma}_{ij}$ are bounded from above and do not decrease during the execution of the algorithm, after a finite number of iterations the flag `continue` has to be equal to 0 at step 26, that is the algorithm stops.

Let us consider now the last iteration of steps 12–25. First, each SaaS j finds the best response to the strategies of the IaaS providers (step 13). Since `continue` has to remain equal to 0 at the last iteration, no SaaS provider modifies the on spot price thresholds $\overline{\sigma}_{ij}$ with respect to the previous iteration. Hence, the optimal strategy found by each IaaS provider at step 25 coincides with the one of the previous iteration. Therefore, the strategy of each player (SaaS/IaaS) is the best response to the ones of the other players, that is the algorithm finds a Generalized Nash Equilibrium. □

Note that, Algorithm 4.1 is suitable of a fully distributed implementation: The SaaS providers initially send to the IaaSs their bid $\overline{\sigma}_{ij}$ and the values of reserved and desired on-spot resources, i.e., r_{ik} and $\overline{s}_{ik}$ obtained solving the ideal scenario problem at step 3. Then, the IaaS providers sends back to individual SaaSs the the number of on spot VMs available s_{ik} (steps 5-10). Then, the best reply procedure starts and messages are exchanged by SaaS providers increasing their bids and IaaS providers. As a final remark, note that Algorithm 4.1 does not require to share the information on the SLA contracts and performance parameters (i.e., $\overline{R}_k$, λ_k, etc.) among SaaSs and IaaSs.

5 Experimental Results

The proposed solution has been evaluated for a variety of systems and workload configurations. Tests have been performed on a VirtualBox virtual machine based on Ubuntu 12.04 server running on an Intel Xeon Nehalem dual socket quad-core system with 32 GB of RAM. CPLEX 12.2.0.0 has been used as MILP solver (IBM ILOG CPLEX Optimizer 2014). Section 5.1 describes the design of experiments. Scalability is discussed in Section 5.2, while Section 5.3 is devoted to quantitatively analyze the efficiency of the equilibria with respect to other approaches proposed in the literature and currently implemented by Cloud providers.

Finally, Section 5.4 illustrates how the SaaSs solution changes by varying the number of target IaaSs, demonstrating cost savings from multi-Cloud adoption.

5.1 Design of Experiments

The proposed approach has been evaluated by considering a very large set of randomly generated instances (800 overall). The performance parameters of the applications and infrastructural resource costs have been randomly generated uniformly in the ranges reported in Table 2 as in other literature approaches (Anselmi and Verloop 2011, Ardagna et al. 2012c, Kusic et al. 2008), considering also real applications (Ardagna et al. 2013) and according to commercial fees applied by IaaS/PaaS Cloud providers (Amazon Elastic Cloud 2014, Microsoft Windows Azure Virtual Machines 2014). Note that VM energy costs include also the overhead of the cooling system by considering the data center PUE, which we assume varies in $[1.2, 1.7]$ according to the values reported in Greenpeace (2012) and Addis et al. (2013).

The upper bounds $\overline{R}_k$ on the average response time were set proportional to the request service demand $1/\mu_{ki}$, i.e., for every SaaS j and application $k \in \mathcal{A}_j$,

$$\overline{R}_k = \gamma_k \min_{i \in \mathcal{I}_j} \frac{1}{\mu_{ki}},$$

where γ_k has been randomly generated uniformly in the range [5, 40], as in Ardagna and Pernici (2007). Since request rejection has an important impact on SaaS provider reputation and are felt by the end users as service unavailability, we set $\lambda_k = 0.8\,\Lambda_k$ (i.e., we assume that at most 20% of request might be rejected) and penalty factors ν_k are setup in a way a request rejection is ten times higher than request execution (i.e., we assume that the SaaS providers rely on the admission control mechanism only in case of resource saturation, while SaaS prefer to pay for the use of resource to support their end users).

Moreover, we analysed the solutions behavior by varying also the ratio of the reserved instances with respect to the total IaaSs resources. In particular, we define the proportion of the available reserved VMs on the total resources offered by IaaS i as

$$\phi_i = \frac{\sum_{j \in \mathcal{S}_i} R_{ij}}{N_i}. \tag{25}$$

Our evaluation is conducted by setting, for all i, $\phi_i = 0.6$ and $\phi_i = 0.7$ (i.e., the SaaSs can rely on the reserved VMs for 60% or 70% of the total IaaS resources). Unfortunately, the data on IaaSs trade policies, i.e., ϕ_i parameters, are not public available.

Table 2: Performance parameters and time unit costs.

N_i	[1000, 2000] VMs	R_{ij}	[100, 200] VMs
D_{ki}	[0.001, 0.05] s	$\overline{R}_k$	[0.025, 0.1] s
μ_{ki}	[200, 400] req/s	η_j	[0.25, 0.75]
Λ_k	[1, 250] req/s	λ_k	[0.8, 200] req/s
ρ_i	[0.048, 0.076] \$/h	ω_i	[0.005, 0.01] \$/h

The number of SaaS providers $|\mathcal{S}|$ has been varied between 100 and 500, the number of applications $|\mathcal{A}|$ (evenly shared among SaaSs) between 1,000 and 5,000[4]. The number of IaaSs $|\mathcal{I}|$ is smaller compared to

[4]We have verified that the performance of Algorithm 4.1 is not affected by the applications to SaaSs assignment cardinality (we varied the number of applications per SaaS in the range 1-100). Results are omitted for space limitation.

SaaSs and it is fixed to 10. This reflects the current business offer. We considered scenarios in which every SaaS relies on three IaaSs at most. All the results reported in the next sections are average values that have been computed on 50 different instances of the same size for every configuration of the Cloud system. Throughout the experiments we set the values $q_{ij} = 0.9$.

5.2 Scalability Analysis

Scalability results are reported in Table 3 which shows average computation and network time, the average number of iterations, and bandwidth requirements for the best reply procedure (steps 12-27 of Algorithm 4.1). We considered the worst case scenario where the cardinality of the IaaSs set is fixed at $|\mathcal{I}| = 10$ and the SaaS to IaaS mapping set cardinality is $|\mathcal{I}_j| = 3$ for all $j \in \mathcal{S}$.

Table 3: Algorithm 4.1 performance statistics.

($\|\mathcal{S}\|$,$\|\mathcal{A}\|$)	Iterations	Comp. Time (s)	Net. Time (s)	IaaS Peak Bandw. (MB/s)	SaaS Peak Bandw. (KB/s)
(100, 1000)	4.38	0.37	2.99	4.42	135.93
(200, 2000)	2.82	0.71	1.92	5.70	87.57
(300, 3000)	4.69	1.08	3.20	14.22	145.65
(400, 4000)	2.39	1.49	1.63	9.68	74.36
(500, 5000)	2.68	4.64	1.83	13.53	83.11

The computational time assumes that at each iteration SaaS problems are solved in parallel (i.e., a distributed implementation is adopted), while the network time considers the time required to access current on spot price and to set the on spot price of running on spot instances on Amazon EC2 (we used the average time measured performing almost 200 tests through the Amazon EC2 python API (Amazon Inc. 2015a) during 22 hours experiment).

The computational time of Algorithm 4.1 increases with the problem instance size, however the average computational time is always below 5 seconds. The network time, which is comparable with the computation time, is proportional to the number of iterations and slightly decreases with the problem instance size. Table 3 reports also the peak bandwidth requirements for SaaSs and IaaSs. SaaS bandwidth is negligible in the order of KB/s. The same observation holds for IaaSs. In the very worst case, peak bandwidth requirements are below 15 MB/s, which is very little for a Cloud data center. The overall Algorithm 4.1 execution time is on average lower than 7s.

Under the assumption to perform run-time resource allocation periodically on a hourly basis (see, e.g., Almeida et al. (2010), Armbrust et al. (2009), Birke et al. (2012)), we can state that the proposed method is very efficient and our solution is suitable to determine the resource provisioning of very large Cloud infrastructures on a hourly basis, without introducing any system overhead.

5.3 Equilibria efficiency

Since the considered problem can admit multiple equilibria, it is worth analyzing the efficiency of the equilibrium found by Algorithm 4.1. In order to evaluate the quality of our approach, we have implemented a different version of Algorithm 4.1, called *Alternative Algorithm*, characterized by a different choice of the initial solution. For each SaaS, the initial number of reserved and desired on spot VMs is determined according to the utilization principle, i.e., the number of VM instances at step 3 is computed such that the average utilization of all VMs, i.e., $x_{ki}/[\mu_{ki}(r_{ki} + \overline{s}_{ki})]$, is equal to $\overline{U}$. Threshold-based approaches have been widely used in the literature (see, e.g., (Wolke and Meixner 2010, Zhu et al. 2009)) and are also advocated by IaaS providers. For example, Amazon Elastic Beanstalk (Amazon Inc. 2014) provides a basic mechanism to

trigger the start-up or termination of VM instances according to the threshold values, which can be specified by SaaS providers accessing the Amazon EC2 API.

In particular, the *Alternative Algorithm* replaces step 3 of Algorithm 4.1 with the following steps:

for $i \in \mathcal{I}_j$ **do**

3a. $x_{ki} = \Lambda_k / |\mathcal{I}_j|, \quad \forall\ k \in \mathcal{A}_j$

3b. $r_{ki} = x_{ki} / (\mu_{ki}\, \overline{U}), \quad \forall\ k \in \mathcal{A}_j$

3c. $\widehat{R}_{ij} = \sum_{k \in \mathcal{A}_j} r_{ki}$

3d. **if** $\widehat{R}_{ij} > R_{ij}$ **then** $r_{ki} = r_{ki}\, R_{ij} / \widehat{R}_{ij}, \quad \forall\ k \in \mathcal{A}_j$

3e. $\overline{s}_{ki} = x_{ki} / (\mu_{ki}\, \overline{U}) - r_{ki}, \quad \forall\ k \in \mathcal{A}_j$

end

In other words, all the incoming workload is served and it is evenly shared among available IaaS (step 3a). Then, the *Alternative Algorithm* tries to use only reserved instances and determines its number according to the utilization threshold (step 3b). If reserved resources are not sufficient (step 3d), reserved instances are rescaled and the number of desired on spot instances is computed (step 3e). As in Ardagna et al. (2012a), Zhu et al. (2009), we set $\overline{U} = 0.6$.

We considered three different metrics to evaluate the efficiency of the two algorithms:

$$OFR_{tot} = \frac{\sum_{j \in \mathcal{S}} \Theta_j \left(GNE^{Alg.\ 4.1}\right)}{\sum_{j \in \mathcal{S}} \Theta_j \left(GNE^{Alternative\ Alg.}\right)},$$

$$OFR_{min} = \min_{j \in \mathcal{S}} \frac{\Theta_j \left(GNE^{Alg.\ 4.1}\right)}{\Theta_j \left(GNE^{Alternative\ Alg.}\right)},$$

$$OFR_{max} = \max_{j \in \mathcal{S}} \frac{\Theta_j \left(GNE^{Alg.\ 4.1}\right)}{\Theta_j \left(GNE^{Alternative\ Alg.}\right)}.$$

OFR_{tot} is the objective function ratio (OFR) between the total SaaS cost at the equilibrium found by Algorithm 4.1 ($GNE^{Alg.\ 4.1}$) and the total SaaS cost at the equilibrium found by the *Alternative Algorithm* ($GNE^{Alternative\ Alg.}$), while OFR_{min} and OFR_{max} are the minimum and maximum ratio between the individual SaaS cost at $GNE^{Alg.\ 4.1}$ and the individual SaaS cost at $GNE^{Alternative\ Alg.}$, respectively. Values lower than 1 indicate that Algorithm 4.1 provides better performance than the *Alternative Algorithm*.

Results in Table 4 show that, on average Algorithm 4.1 provides solutions better than 5% of the alternative one. If the individual SaaS are considered, OFR_{min} ratio ranges between 38% and 52%, that is in the worst case the *Alternative Algorithm* provides a solution worst than 192–261%, while Algorithm 4.1 always performs at least as the Alternative one (OFR_{max} is always equal to 1). The last column of Table 4 reports the percentage value of the number of SaaS providers such that Algorithm 4.1 performs better than the alternative one, i.e., $\Theta_j \left(GNE^{Alg.\ 4.1}\right) < \Theta_j \left(GNE^{Alternative\ Alg.}\right)$.

Table 4: Alternative Algorithm Comparison.

$(\lvert\mathcal{S}\rvert, \lvert\mathcal{A}\rvert)$	OFR_{tot}	OFR_{min}	OFR_{max}	% SaaS performing better
(100, 1000)	0.9413	0.3917	1.0	11.08
(200, 2000)	0.9527	0.5201	1.0	7.62
(300, 3000)	0.9556	0.3923	1.0	7.13
(400, 4000)	0.9416	0.3813	1.0	9.12
(500, 5000)	0.9429	0.4722	1.0	8.58

5.4 Multiple IaaS Analysis

In this Section, we consider SaaS providers relying on one or more IaaS providers to host their applications in order to highlight the benefits of the adoption of multi-IaaS solutions. In particular for every SaaS j, the number of target IaaSs $|\mathcal{I}_j|$ has been varied between 1 and 3. In this Section we always consider $|\mathcal{S}| = 300$ and $|\mathcal{A}| = 3,000$.

To perform a fair comparison, the total number of reserved VMs available is always the same both in the single IaaS case and in the multi-IaaS scenario. Problem instances have been randomly generated such that

$$\sum_{i \in \mathcal{I}_j,\, |\mathcal{I}_j|=1} R_{ij} = \sum_{i \in \mathcal{I}_j,\, |\mathcal{I}_j|=2} R_{ij} = \sum_{i \in \mathcal{I}_j,\, |\mathcal{I}_j|=3} R_{ij}, \qquad \forall\ j \in \mathcal{S},$$

where also the arrival rates Λ_k are fixed. Results are shown in Table 5, where the average SaaS costs at the equilibrium are expressed in \$.

Table 5: Average SaaS cost (\$) for Multi-IaaS adoption.

ϕ_i	$\lvert\mathcal{I}_j\rvert = 1$	$\lvert\mathcal{I}_j\rvert = 2$	$\lvert\mathcal{I}_j\rvert = 3$
0.6	15.5489	14.2971	13.1112
0.7	16.2184	14.5385	14.2297

The average SaaS cost decreases as the cardinality of $\mathcal{I}_j$ increases both for $\phi_i = 0.6$ and $\phi_i = 0.7$ (i.e., the SaaSs can rely on the reserved VMs for 60% or 70% of the total IaaS resources). This suggests an effective advantage of having more IaaS providers from the SaaS point of view. Using two IaaS, savings range in 8–10%, while adopting three IaaS providers saving grow up to 12–15% with respect to the single IaaS scenario.

This could be expected, since with multiple IaaS providers when resources offered by a IaaS are saturated, SaaSs can start VMs into another IaaS provider without incurring in penalties for request rejections. Moreover, with the possibility of allocating resources in more than one IaaS, SaaSs can compete for the cheapest resources on multiple providers, in order to reduce their fees. Indeed, a SaaS can bid for lowest on spot price resources, decreasing its cost thresholds $\overline{\sigma}_{ij}$ according to the different prices σ_i.

Finally, we computed the average on spot prices σ_i fixed by IaaSs for the three cases: values are reported in Table 6.

Results do not vary significantly. Since the number of reserved instances is the same across different scenarios, we can argue that the SaaSs savings are due to the fact that SaaSs can access resources from the cheapest provider.

Table 6: Average on spot prices σ_i ($).

ϕ_i	$\lvert\mathcal{I}_j\rvert = 1$	$\lvert\mathcal{I}_j\rvert = 2$	$\lvert\mathcal{I}_j\rvert = 3$
0.6	0.7346	0.7490	0.7515
0.7	0.7362	0.7226	0.7463

In conclusion, our analysis demonstrates that using simultaneously multiple IaaS providers allows both to improve the availability for the SaaS end-users and to achieve economic benefits.

6 Conclusions

In this paper, we considered the problem of run-time management of IaaSs provider capacities among multiple competing SaaSs through the formulation and study of a GNE model. We took the perspective of SaaS providers whose goal is the minimization of the costs associated with the virtual machine instances allocated on multiple IaaSs, while guaranteeing QoS constraints. On the other side, each IaaS provider aims at maximizing its revenues. The cost model includes SaaS revenues and penalties incurred for request rejections and infrastructural costs associated with IaaSs resources. Current on spot pricing models adopted by IaaS providers are considered, which lead to a nonconvex generalized Nash game for which current literature results cannot guarantee even the equilibrium existence.

We proposed a solution technique based on the best reply dynamics and evaluated the effectiveness of our approach, performing a wide set of analyses which considered multiple workloads and system configurations. Scalability analysis have shown that systems up to thousands of applications can be managed very efficiently in a fully distributed manner. Since the execution (including computation and network) times required to solve problem instances of maximum size were around 7 seconds in the worst case, we can state that our approach can be adopted on a hourly basis, without introducing any system overhead. When compared with an alternative method inspired by other literature solutions and currently implemented by IaaS providers, significant cost savings can be achieved (5% on average, up to 260% for individual SaaS providers). Furthermore, we have shown that varying the number of IaaS providers 8-15% cost savings can be achieved from the workload distribution on multiple IaaSs.

Future work will be devoted to a deeper investigation of the time scales which can be adopted to govern the behavior of Cloud systems with a more fine grained modelling of servers energy consumption, performing resource allocation also every few minutes. Other approaches are worthy of being studied and developed in order to compare different equilibria. Furthermore, on demand resources will be included in the game formulation and the model will be extended to provide availability guarantees. Finally, systems supporting also batch applications (e.g., map-reduce and business intelligence) will be also considered.

Acknowledgment

Danilo Ardagna's work is partially supported by the European Commission grant no. FP7-ICT-2011-8-318484 (MODAClouds).

References

Abhishek, V., I.A. Kash, P. Key. 2012. Fixed and market pricing for cloud services. *Computer Communications Workshops (INFOCOM WKSHPS), 2012 IEEE Conference on*. 157–162.

Addis, B., D. Ardagna, B. Panicucci, M.S. Squillante, Li Zhang. 2013. A hierarchical approach for the resource management of very large cloud platforms. *IEEE Trans. Dependable Secure Comput.* **10** 253–272.

Almeida, J., V. Almeida, D. Ardagna, I. Cunha, C. Francalanci, M. Trubian. 2010. Joint admission control and resource allocation in virtualized servers. *J. Parallel Distr. Com.* **70** 344–362.

Altman, E., T. Boulogne, R. El Azouzi, T. Jiménez, L. Wynter. 2006. A survey on networking games in telecommunications. *Comput. Oper. Res.* **33** 286–311.

Amazon Elastic Cloud. 2014. `http://aws.amazon.com/ec2/`.

Amazon Inc. 2014. AWS Elastic Beanstalk. `http://aws.amazon.com/elasticbeanstalk/`.

Amazon Inc. 2015a. AWS SDK for Python (Boto). `http://aws.amazon.com/sdk-for-python/`.

Amazon Inc. 2015b. Elastic Load Balancing. `http://aws.amazon.com/elasticloadbalancing/`.

Amazon Web Services. 2014. `http://aws.amazon.com/`.

Anselmi, J., D. Ardagna, M. Passacantando. 2014. Generalized Nash Equilibria for SaaS/PaaS Clouds. *Eur. J. Oper. Res.* **236** 326–339.

Anselmi, J., I.M. Verloop. 2011. Energy-aware capacity scaling in virtualized environments with performance guarantees. *Perform. Evaluation* **68** 1207–1221.

Ardagna, D., S. Casolari, M. Colajanni, B. Panicucci. 2012a. Dual time-scale distributed capacity allocation and load redirect algorithms for cloud systems. *J. Parallel Distr. Com.* **72** 796–808.

Ardagna, D., E. di Nitto, P. Mohagheghi, S. Mosser, C. Ballagny, F. D'Andria, G. Casale, P. Matthews, C.-S. Nechifor, D. Petcu, A. Gericke, C. Sheridan. 2012b. Modaclouds: A model-driven approach for the design and execution of applications on multiple clouds. *Modeling in Software Engineering (MISE), 2012 ICSE Workshop on*. 50–56.

Ardagna, D., B. Panicucci, M. Passacantando. 2011. A game theoretic formulation of the service provisioning problem in cloud systems. Sadagopan Srinivasan, Krithi Ramamritham, Arun Kumar, M. P. Ravindra, Elisa Bertino, Ravi Kumar, eds., *WWW*. ACM, 177–186.

Ardagna, D., B. Panicucci, M. Passacantando. 2013. Generalized Nash equilibria for the service provisioning problem in cloud systems. *IEEE Trans. Serv. Comput.* **6** 429–442.

Ardagna, D., B. Panicucci, M. Trubian, L. Zhang. 2012c. Energy-aware autonomic resource allocation in multitier virtualized environments. *IEEE Trans. Serv. Comput.* **5** 2–19.

Ardagna, D., B. Pernici. 2007. Adaptive service composition in flexible processes. *IEEE Trans. Software Eng.* **33** 369 –384.

Armbrust, M., A. Fox, R. Griffith, A. D. Joseph, R. H. Katz, A. Konwinski, G. Lee, D. A. Patterson, A. Rabkin, I. Stoica, M. Zaharia. 2009. Above the clouds: A Berkeley view of cloud computing. Tech. Rep. UCB/EECS-2009-28, EECS Department, University of California, Berkeley.

Bigi, G., M. Castellani, M. Pappalardo, M. Passacantando. 2013. Existence and solution methods for equilibria. *Eur. J. Oper. Res.* **227** 1–11.

Birke, R., L.Y. Chen, E. Smirni. 2012. Data centers in the cloud: A large scale performance study. *Cloud Computing (CLOUD), 2012 IEEE 5th International Conference on*. 336–343.

Cavazzuti, E., M. Pappalardo, M. Passacantando. 2002. Nash equilibria, variational inequalities, and dynamical systems. *J. Optim. Theory Appl.* **114** 491–506.

Debreu, G. 1952. A social equilibrium existence theorem. *Proc. Nat. Acad. Sci. U.S.A.* **38** 886–893.

Dube, P., Z. Liu, L. Wynter, C. H. Xia. 2007. Competitive equilibrium in e-commerce: Pricing and outsourcing. *Comput. Oper. Res.* **34** 3541–3559.

Facchinei, F., C. Kanzow. 2010a. Generalized Nash equilibrium problems. *Ann. Oper. Res.* **175** 177–211.

Facchinei, F., C. Kanzow. 2010b. Penalty methods for the solution of generalized Nash equilibrium problems. *SIAM J. Optim.* **20** 2228–2253.

Feng, Y., B. Li, B. Li. 2014. Price competition in an oligopoly market with multiple iaas cloud providers. *IEEE Trans. Comput.* **63** 59–73.

Gohad, A., N.C. Narendra, P. Ramachandran. 2013. Cloud pricing models: A survey and position paper. *Cloud Computing in Emerging Markets (CCEM), 2013 IEEE International Conference on*. 1–8.

Google Compute Engine. 2014. `https://cloud.google.com/products/compute-engine/`.

Greenpeace. 2012. How Clean is your Cloud? `http://www.greenpeace.org/international/Global/international/publications/climate/2012/iCoal/HowCleanisYourCloud.pdf`.

IBM ILOG CPLEX Optimizer. 2014. `http://www-01.ibm.com/software/integration/optimization/cplex-optimizer/`.

Kumar, D., A. Tantawi, Li Zhang. 2009. Real-time performance modeling for adaptive software systems with multi-class workload. *Modeling, Analysis Simulation of Computer and Telecommunication Systems, 2009. MASCOTS '09. IEEE International Symposium on*. 1–4.

Kusic, D., J.O. Kephart, J.E. Hanson, N. Kandasamy, G. Jiang. 2008. Power and performance management of virtualized computing environments via lookahead control. *Autonomic Computing, 2008. ICAC '08. International Conference on*. 3–12.

Menache, I., A. Ozdaglar, N. Shimkin. 2011. Socially optimal pricing of cloud computing resources. *Proceedings of the 5th International ICST Conference on Performance Evaluation Methodologies and Tools*. VALUETOOLS '11, ICST (Institute for Computer Sciences, Social-Informatics and Telecommunications Engineering), ICST, Brussels, Belgium, Belgium, 322–331.

Menasc, Daniel A., Mohamed N. Bennani. 2006. Analytic performance models for single class and multiple class multithreaded software servers. *Int. CMG Conference'06*. 475–482.

Microsoft Windows Azure. 2014. `http://www.windowsazure.com/`.

Microsoft Windows Azure Virtual Machines. 2014. `http://www.windowsazure.com/en-us/home/features/virtual-machines/`.

MODAClouds. 2014. `http://www.modaclouds.eu/`.

Nabetani, K., P. Tseng, M. Fukushima. 2011. Parametrized variational inequality approaches to generalized Nash equilibrium problems withshared constraints. *Comput. Optim. Appl.* **48** 423–452.

Nash, J. 1951. Non-cooperative games. *Ann. of Math.* **54** 286–295.

Pang, J.-S., M. Fukushima. 2005. Quasi-variational inequalities, generalized Nash equilibria, and multi-leader-follower games. *Comput. Manag. Sci.* **2** 21–56.

Panicucci, B., M. Pappalardo, M. Passacantando. 2009. On solving generalized Nash equilibrium problems via optimization. *Optim. Lett.* **3** 419–435.

Roh, Heejun, Cheoulhoon Jung, Wonjun Lee, Ding-Zhu Du. 2013. Resource pricing game in geo-distributed clouds. *INFOCOM, 2013 Proceedings IEEE*. 1519–1527.

Rosen, J. B. 1965. Existence and uniqueness of equilibrium points for concave n-person games. *Econometrica* **33** 520–534.

Song, Yang, M. Zafer, Kang-Won Lee. 2012. Optimal bidding in spot instance market. *INFOCOM, 2012 Proceedings IEEE*. 190–198.

Urgaonkar, Bhuvan, Giovanni Pacifici, Prashant Shenoy, Mike Spreitzer, Asser Tantawi. 2007. Analytic modeling of multitier internet applications. *ACM Trans. Web* **1**.

von Heusinger, A., C. Kanzow. 2009. Relaxation methods for generalized Nash equilibrium problems with inexact line search. *J. Optim. Theory Appl.* **143** 159–183.

von Heusinger, A., C. Kanzow, M. Fukushima. 2012. Newton's method for computing a normalized equilibrium in the generalized Nash game through fixed point formulation. *Math. Program.* **132** 99–123.

Wan, Jian, Dechuan Deng, Congfeng Jiang. 2012. Non-cooperative gaming and bidding model based resource allocation in virtual machine environment. *Parallel and Distributed Processing Symposium Workshops PhD Forum (IPDPSW), 2012 IEEE 26th International*. 2183–2188.

Wei, G., A. V. Vasilakos, Y. Zheng, N. Xiong. 2010. A game-theoretic method of fair resource allocation for cloud computing services. *J. Supercomput.* **54** 252–269.

Wolke, Andreas, Gerhard Meixner. 2010. Twospot: A cloud platform for scaling out web applications dynamically. Elisabetta Di Nitto, Ramin Yahyapour, eds., *Towards a Service-Based Internet*, *Lecture Notes in Computer Science*, vol. 6481. Springer Berlin Heidelberg, 13–24.

Zafer, M., Yang Song, Kang-Won Lee. 2012. Optimal bids for spot vms in a cloud for deadline constrained jobs. *Cloud Computing (CLOUD), 2012 IEEE 5th International Conference on*. 75–82.

Zhang, L., X. Meng, S. Meng, J. Tan. 2013. K-scope: Online performance tracking for dynamic cloud applications. *Presented as part of the 10th International Conference on Autonomic Computing*. USENIX, Berkeley, CA, 29–32.

Zhang, Q., Q. Zhu, M. F. Zhani, R. Boutaba. 2012. Dynamic service placement in geographically distributed clouds. *Distributed Computing Systems (ICDCS), 2012 IEEE 32nd International Conference on*. 526–535.

Zhu, Xiaoyun, Donald Young, Brian J. Watson, Zhikui Wang, Jerry Rolia, Sharad Singhal, Bret Mckee, Chris Hyser, Daniel Gmach, Robert Gardner, Tom Christian, Ludmila Cherkasova. 2009. 1000 islands: An integrated approach to resource management for virtualized data centers. *Cluster Comput.* **12** 45–57.